



Chapter Four: Loops II

Chapter Goals

- To understand nested loops
- To implement programs that read and process data sets
- To use a computer for simulations

Processing Input – When and/or How to Stop?



or be stopped!

Processing Input – When and/or How to Stop?

- We need to know, when getting input from a user, when they are done.
- One method is to hire a sentinel (as shown) or more correctly choose a *value* whose meaning is STOP!
- In the military, a sentinel guards a border or passage. In computer science, a sentinel value denotes the end of an input sequence or the border between input sequences.
- As long as there is a known range of valid data points, we can use a value not in it.



Processing Input – When and/or How to Stop?

- We will write code to calculate the average of some salary values input by the user.

How many will there be?

That is the problem. We can't know.

But we can use a *sentinel value*, as long as we tell the user to use it, to tell us when they are done.

- Since salaries are never negative, we can safely choose -1 as our sentinel value.

Processing Input – When and/or How to Stop?

- In order to have a value to test, we will need to get the first input before the loop. The loop statements will process each non-sentinel value, and then get the next input.
- Suppose the user entered the sentinel value as the first input. Because averages involve division by the count of the inputs, we need to protect against dividing by zero. Using an **if-else** statement from Chapter 3 will do.

The Complete Salary Average Program

```
#include <iostream>
using namespace std;
```

ch04/sentinel.cpp

```
int main()
{
    double sum = 0;
    int count = 0;
    double salary = 0;
    // get all the inputs
    cout << "Enter salaries, -1 to finish: ";
    while (salary != -1)
    {
        cin >> salary;
        if (salary != -1)
        {
            sum = sum + salary;
            count++;
        }
    }
}
```

The Complete Salary Average Program

```
// process and display the average
if (count > 0)
{
    double average = sum / count;
    cout << "Average salary: " << average << endl;
}
else
{
    cout << "No data" << endl;
}

return 0;
}
```

A program run:

```
Enter salaries, -1 to finish: 10 10 40 -1
Average salary: 20
```


Using Failed Input for Processing

- Sometimes it is easier and a bit more intuitive to ask the user to “Hit Q to Quit” instead of requiring the input of a sentinel value.
- Sometimes picking a sentinel value is simply impossible – if any valid number is allowed, which number could be chosen?

Using Failed Input for Processing

- In the previous chapter, we used `cin.fail()` to test if the most recent input failed.
- Note that if you intend to take more input from the keyboard after using failed input to end a loop, you must reset the keyboard with `cin.clear()`.

Using Failed Input for Processing

If we introduce a bool variable to be used to test for a failed input, we can use `cin.fail()` to test for the input of a 'Q' when we were expecting a number:

Using Failed Input for Processing

```
cout << "Enter values, Q to quit: ";
bool more = true;
while (more)
{
    cin >> value;
    if (cin.fail())
    {
        more = false;
    }
    else
    {
        // process value here
    }
}
cin.clear() // reset if more input is to be taken
```

Using Failed Input for Processing

- Using a `bool` variable in this way is disliked by many programmers.

Why?

- `cin.fail` is set *when* `>>` fails
It is not really a top or bottom test.

If only we could use the input itself to control the loop – we can!

- An input that does not succeed is considered to be `false` so it can be used in the `while`'s test.

Using Failed Input for Processing

Using the input attempt directly we have:

```
cout << "Enter values, Q to quit: ";  
while (cin >> value)  
{  
    // process value here  
}  
cin.clear();
```

The Loop and a Half Problem and the `break` Statement

Those same programmers who dislike loops that are controlled by a `bool` variable have another reason: the actual test for loop termination is in the *middle* of the loop. Again it is not really a top or bottom test.

This is called a loop-and-a-half.

The Loop and a Half Problem and the `break` Statement

If we test for a failed read, we can stop the loop *at that point*:

```
while (true)
{
    cin >> value;
    if (cin.fail()) { break; }
    // process value here
}
cin.clear() // reset if more input is to be taken
```

The **break** statement breaks out of the enclosing loop, independent of the loop condition.

Nested Loops



For each hour, 60 minutes are processed – a nested loop.

Nested Loops

- Nested loops are used mostly for data in tables as rows and columns.
- The processing across the columns is a loop, as you have seen before, “nested” inside a loop for going down the rows.
- Each row is processed similarly so design begins at that level. After writing a loop to process a generalized row, that loop, called the “inner loop,” is placed inside an “outer loop.”

Nested Loops

Write a program to produce a table of powers.
The output should be something like this:

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
...
10	100	1000	10000

Nested Loops

- The first step is to solve the “nested” loop.
- There are four columns and in each column we display the power. Using x to be the number of the row we are processing, we have (in pseudo-code):

```
for n from 1 to 4
{
    print  $x^n$ 
}
```

- You would test that this works in your code before continuing. If you can't correctly print one row, why try printing lots of them?

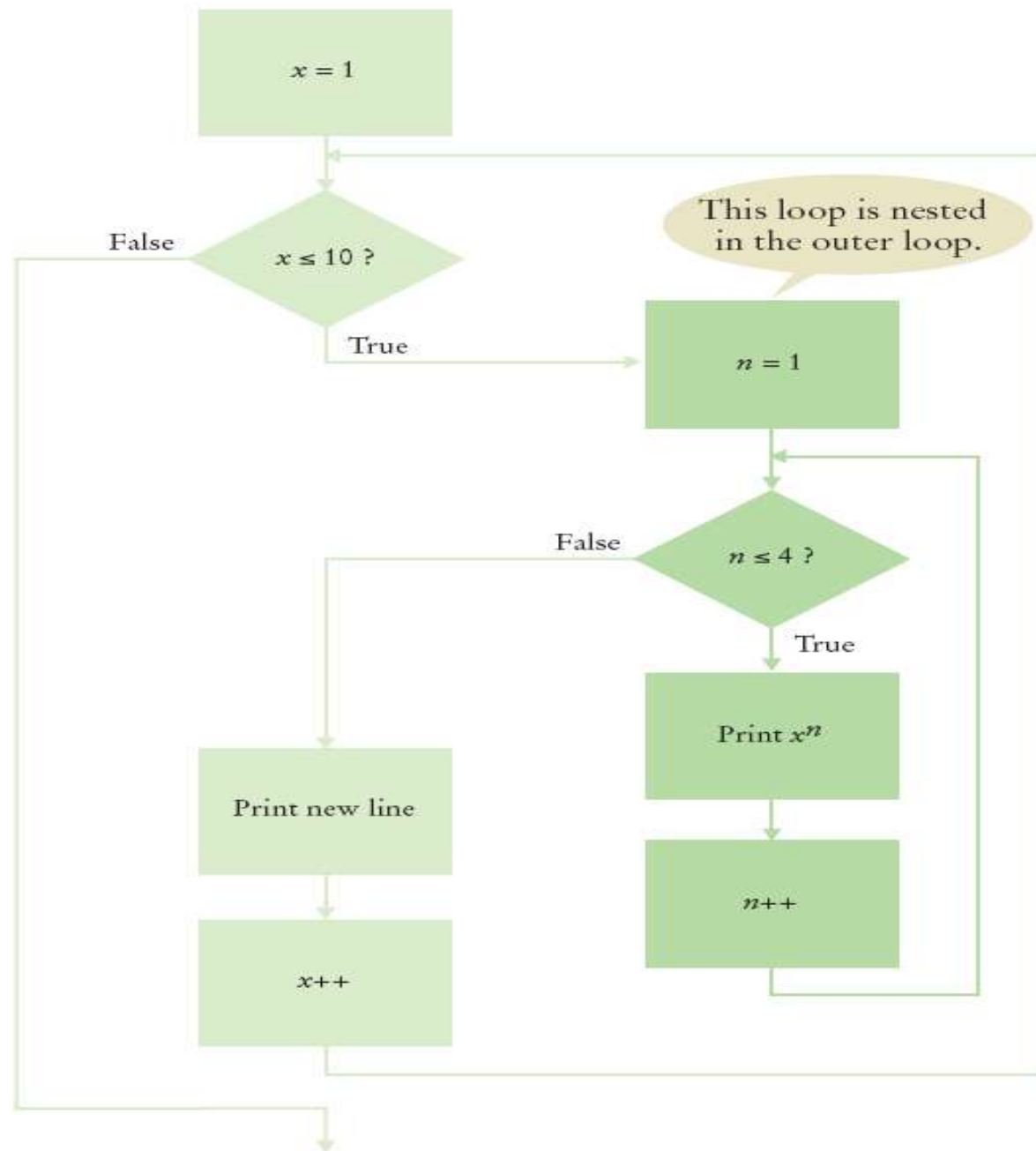
Nested Loops

Now, putting the inner loop into the whole process we have:

(don't forget to indent, nestedly)

```
print table header
for x from 1 to 10
{
    print table row
    print endl
}
```

Nested Loops



The Complete Program for Table of Powers

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

int main()
{
    const int NMAX = 4;
    const double XMAX = 10;

    // Print table header
    for (int n = 1; n <= NMAX; n++)
    {
        cout << setw(10) << n;
    }
    cout << endl;
    for (int n = 1; n <= NMAX; n++)
    {
        cout << setw(10) << "x ";
    }
    cout << endl << endl;
}
```

ch04/powtable.cpp

The Complete Program for Table of Powers

```
// Print table body
for (double x = 1; x <= XMAX; x++)
{
    // Print table row
    for (int n = 1; n <= NMAX; n++)
    {
        cout << setw(10) << pow(x, n) ;
    }
    cout << endl;
}

return 0;
}
```

The program run would be:

1	2	3	4
x	x	x	x
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625

More Nested Loop Examples

The loop variables can have a value relationship. In this example the inner loop depends on the value of the outer loop.

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        cout << "*";  
cout << endl;
```

The output will be:

```
*  
**  
***  
****
```

More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        cout << "*" ;  
cout << endl ;
```

j is *each* line's length,
which is different for each line and
depends on the current line number, *i*

i represents the
row number or
the line number

```
*  
* *  
* * *  
* * * *
```

More Nested Loop Examples

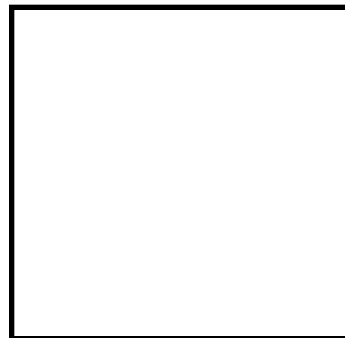
```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        cout << "*";  
cout << endl;
```

j is *each* line's length,
which is different for each line and
depends on the current line number, *i*

j stops at: *i*
1

when *i* is: *i* 1

i represents the
row number or
the line number



More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        cout << "*";  
cout << endl;
```

j is *each* line's length,
which is different for each line and
depends on the current line number, *i*

j stops at: *i*
1

when *i* is: *i* 1

*

i represents the
row number or
the line number

More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        cout << "*";  
cout << endl;
```

j is *each* line's length,
which is different for each line. and
depends on the current line number, *i*

j stops at: *i*
1

when *i* is: *i* 1 *
 i 2 * *

i represents the
row number or
the line number

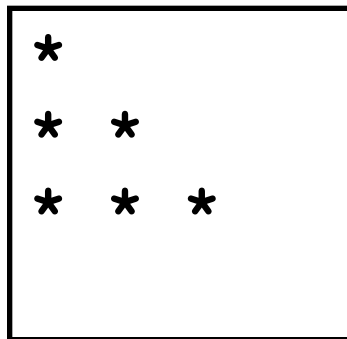
More Nested Loop Examples

```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        cout << "*";  
cout << endl;
```

j is *each* line's length,
which is different for each line and
depends on the current line number, i

j stops at: i i i
 1 2 3

when i is: i 1
 i 2
 i 3



i represents the
row number or
the line number

More Nested Loop Examples

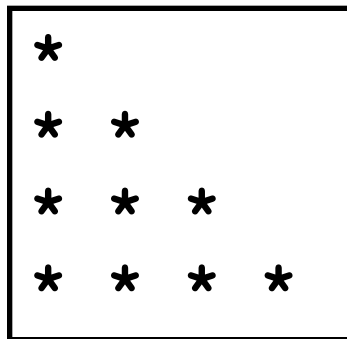
```
for (i = 1; i <= 4; i++)  
    for (j = 1; j <= i; j++)  
        cout << "*";  
cout << endl;
```

j is *each* line's length,
which is different for each line and
depends on the current line number, i

j stops at: i i i i
 1 2 3 4

when i is: i 1
 i 2
 i 3
 i 4

i represents the
row number or
the line number



More Nested Loop Examples

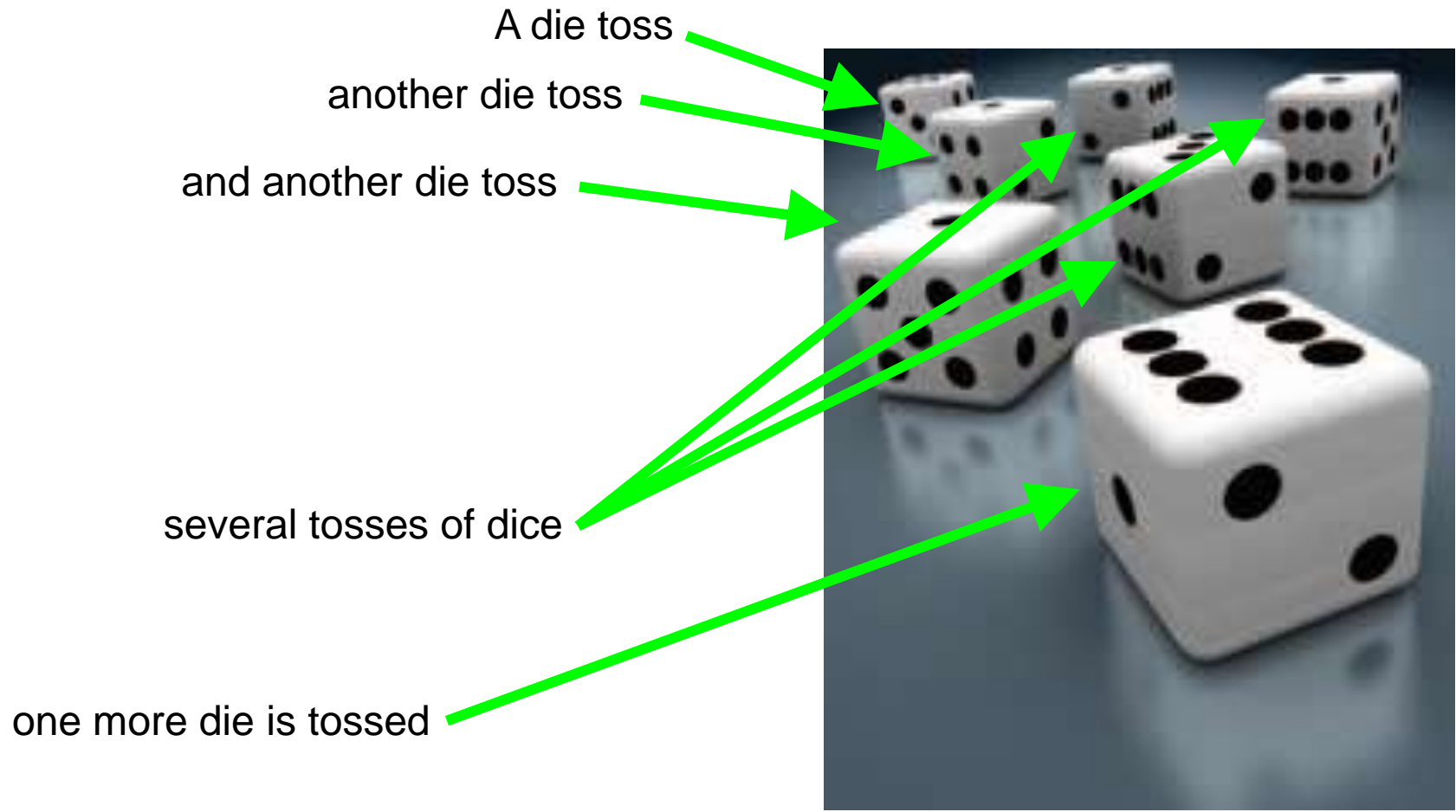
In this example, the loop variables are still related, but the processing is a bit more complicated.

```
for (i = 1; i <= 3; i++)
{
    for (j = 1; j <= 5; j++)
    {
        if (i + j % 2 == 0) { cout << "*"; }
        else { cout << " "; }
    }
    cout << endl;
}
```

The output will be:

```
* * *
 * *
* * *
```


Random Numbers and Simulations



was that an English lesson?

Simulations

A simulation program uses the computer to simulate an activity in the real world (or in an imaginary one).

Simulations

- Simulations are commonly used for
 - Predicting climate change
 - Analyzing traffic
 - Picking stocks
 - Many other applications in science and business

Randomness for Reality (Simulating)

- Programmers must model the “real world” at times.
- Consider the problem of modeling customers arriving at a store.

Do we know the rate?

Does anyone?

How about the shopkeeper!

Randomness for Reality (Simulating)

Ask the shopkeeper:

*It's about every five minutes
...or so...
...give or a take a couple...
...or three...
...but on certain Tuesdays...*



©Silberander, EthneGraphics®

Randomness for Reality (Simulating)

To accurately model customer traffic, you want to take that random fluctuation into account.

How?

The rand Function

The C++ library has a random number generator:

`rand()`

The `rand` Function

`rand` is defined in the `cstdlib` header

Calling `rand` yields a random integer
between 0 and `RAND_MAX`

(The value of `RAND_MAX` is implementation dependent)

The `rand` Function

Calling `rand` again yields a different random integer

Very, very, very rarely it might be the same random integer again.

(That's OK. In the real world this happens.)

The `rand` Function

`rand` picks from a very long sequence of numbers that don't repeat for a long time.

But they do eventually repeat.

These sorts of “random” numbers are often called *pseudorandom numbers*.

The `rand` Function

`rand` uses only one pseudorandom number sequence and it always starts from the same place.

Oh dear

The `rand` Function

When you run your program again on another day, the call to `rand` will start with:

the same random number!

Is it very “real world” to use the same sequence over and over?

No, but it’s really nice for testing purposes.

but...

Seeding the `rand` Function

You can “seed” the random generator to indicate where it should start in the pseudorandom sequence

Calling `srand` sets where `rand` starts

`srand` is defined in the `cstdlib` header

Seeding the `rand` Function

But what value would be different every *time* you run your program?

(hint)

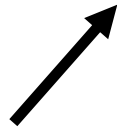


How about the time?

Seeding the `rand` Function

You can obtain the system time.

Calling `time(0)` gets the current time



Note the zero. It is required.

`time` is defined in the `time` header

Seeding the `rand` Function

Calling `srand` sets where `rand` starts.

Calling `time (0)` gets the current time.

So, to set up for “really, really random”
random numbers on each program run:

```
srand(time(0)); // seed rand()
```

(Well, as “really random” as we can hope for.)

Modeling Using the `rand` Function

Let's model a pair of dice,



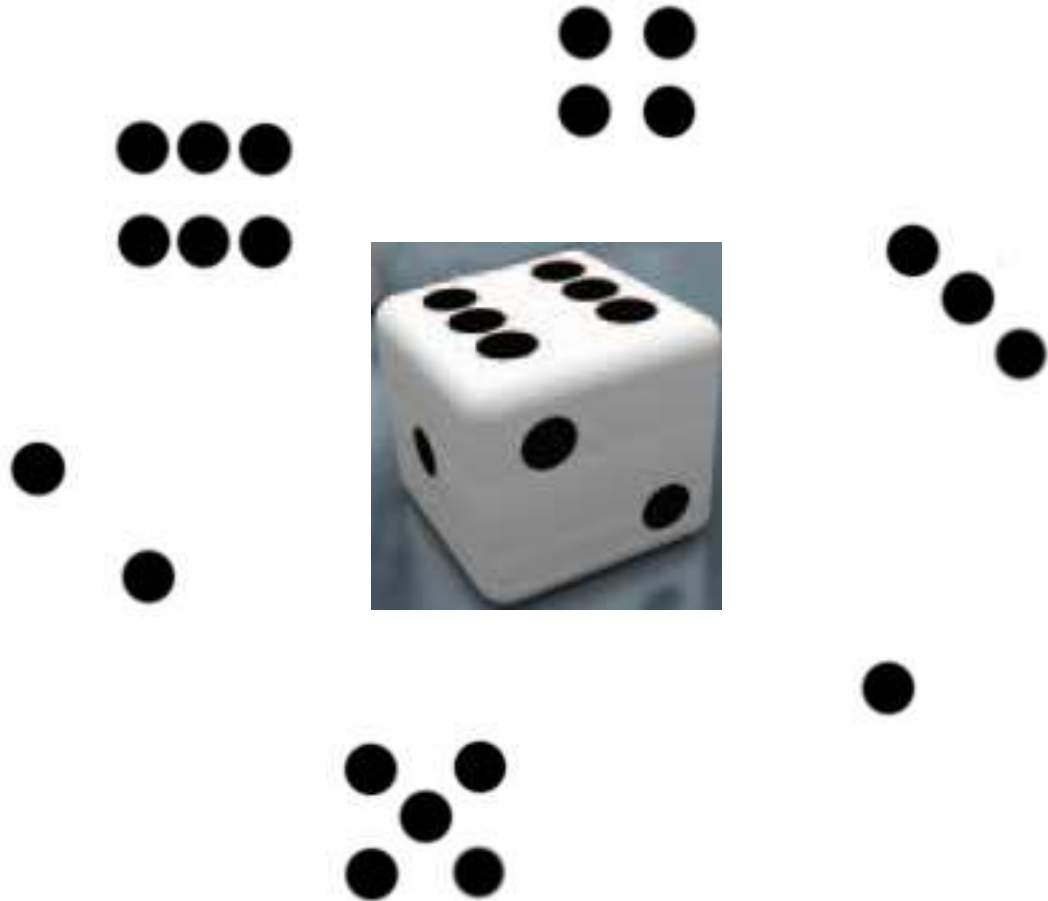
Modeling Using the `rand` Function



one die at a time.

Modeling Using the `rand` Function

What are the numbers on one die?



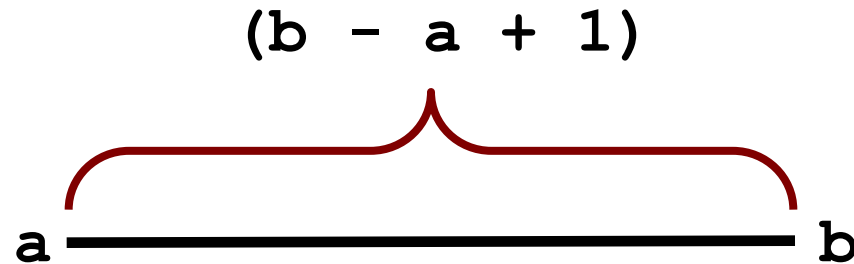
Modeling Using the `rand` Function

What are the bounds of the range of numbers on one die?
1 and 6 (inclusive)



We want a value randomly between those endpoints
(inclusively)

Modeling Using the `rand` Function

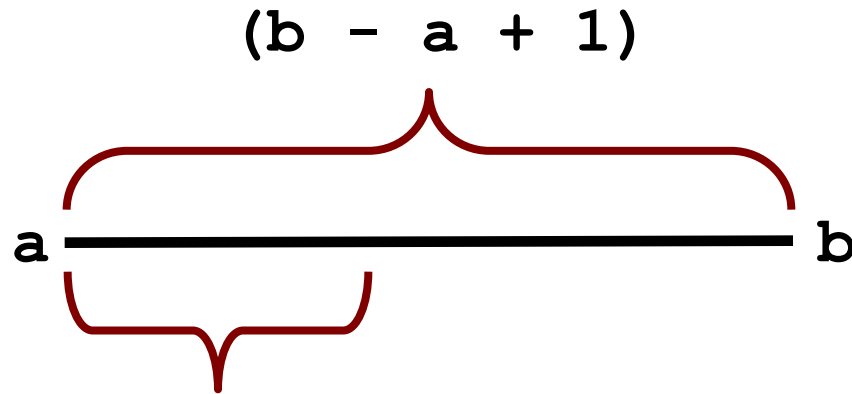


Given two endpoints,
a and **b**, recall there are

$$(b - a + 1)$$

values between **a** and **b**,
(including the bounds themselves).

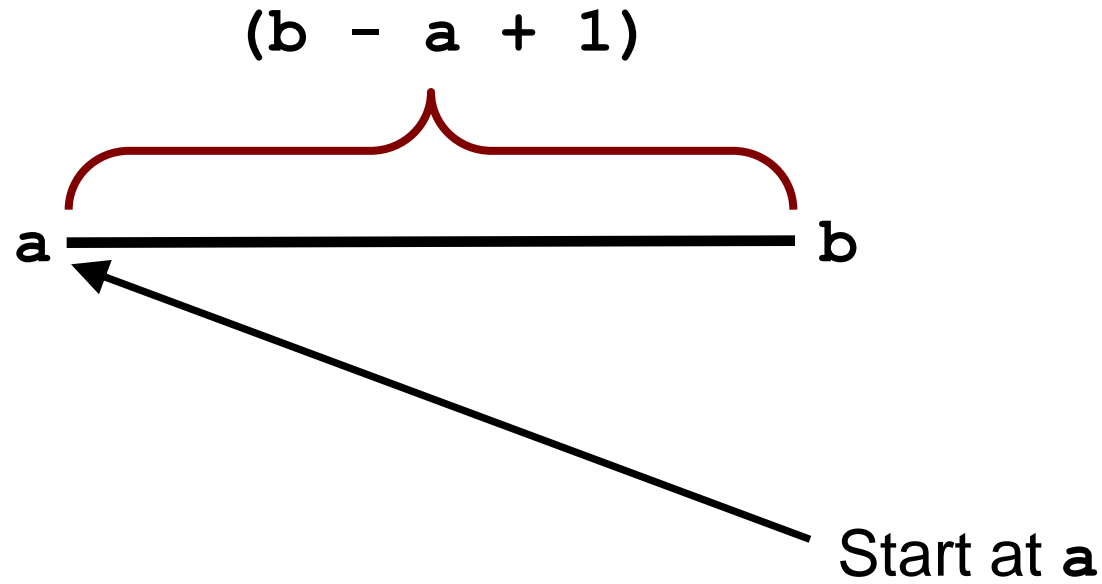
Modeling Using the `rand` Function



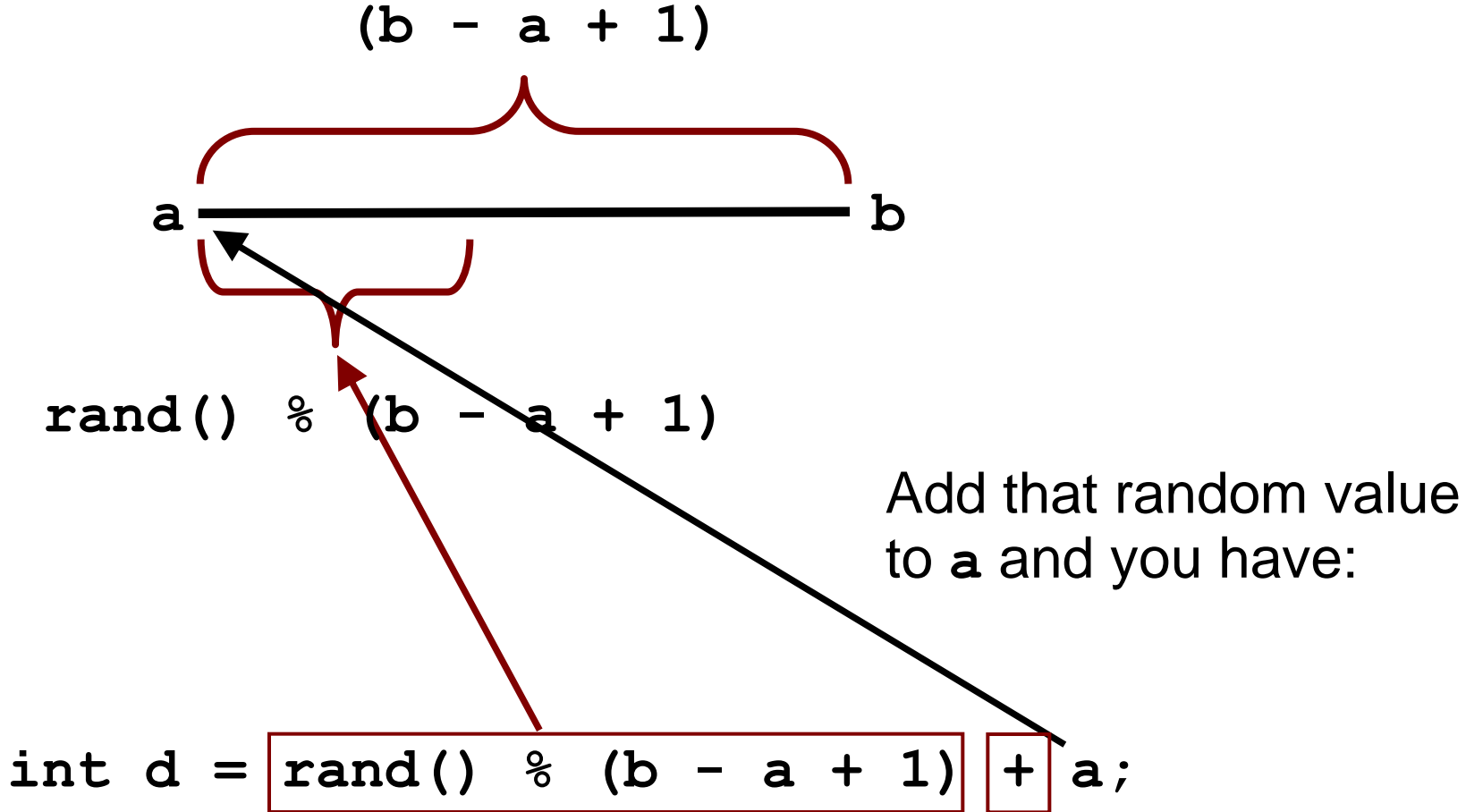
`rand() % (b - a + 1)`

Obtain a random value
between 0 and `b - a`
by using the `rand()` function

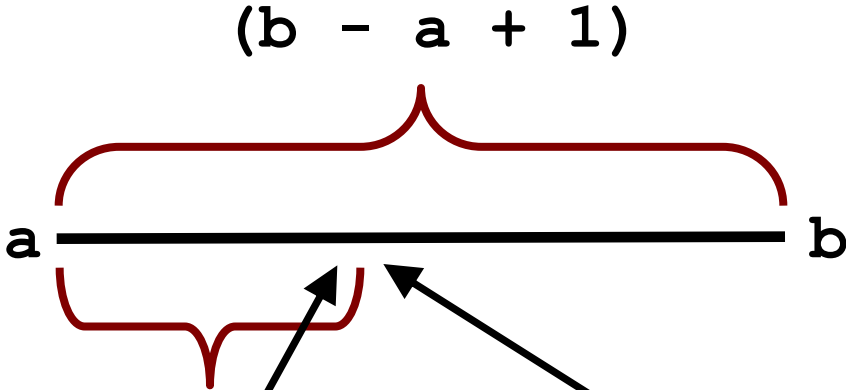
Modeling Using the `rand` Function



Modeling Using the rand Function



Modeling Using the rand Function



a random value in the range.

```
int d = rand() % (b - a + 1) + a;
```

Modeling Using the `rand` Function



Using 1 and 6 as the bounds
and
modeling for two dice,
running for 10 tries,

we have:

Modeling Using the rand Function

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
    srand(time(0));

    for (i = 1; i <= 10; i++)
    {
        int d1 = rand() % 6 + 1;
        int d2 = rand() % 6 + 1;
        cout << d1 << " " << d2 << endl;
    }
    cout << endl;
    return 0;
}
```

ch04/dice.cpp

One of many different
Program Runs:

```
5 1
2 1
1 2
5 1
1 2
6 4
4 4
6 1
6 3
5 2
```

The Monte Carlo Method



The premier gaming “*table d’darts*”
at one of the less well known casinos in Monte Carlo,
somewhat close but not quite next door to Le Grand Casino.

The Monte Carlo Method



As long as we're here, let's go in!

The Monte Carlo Method

The Monte Carlo method is a method for finding approximate solutions to problems that cannot be precisely solved.

Here is an example: compute π

This is difficult.

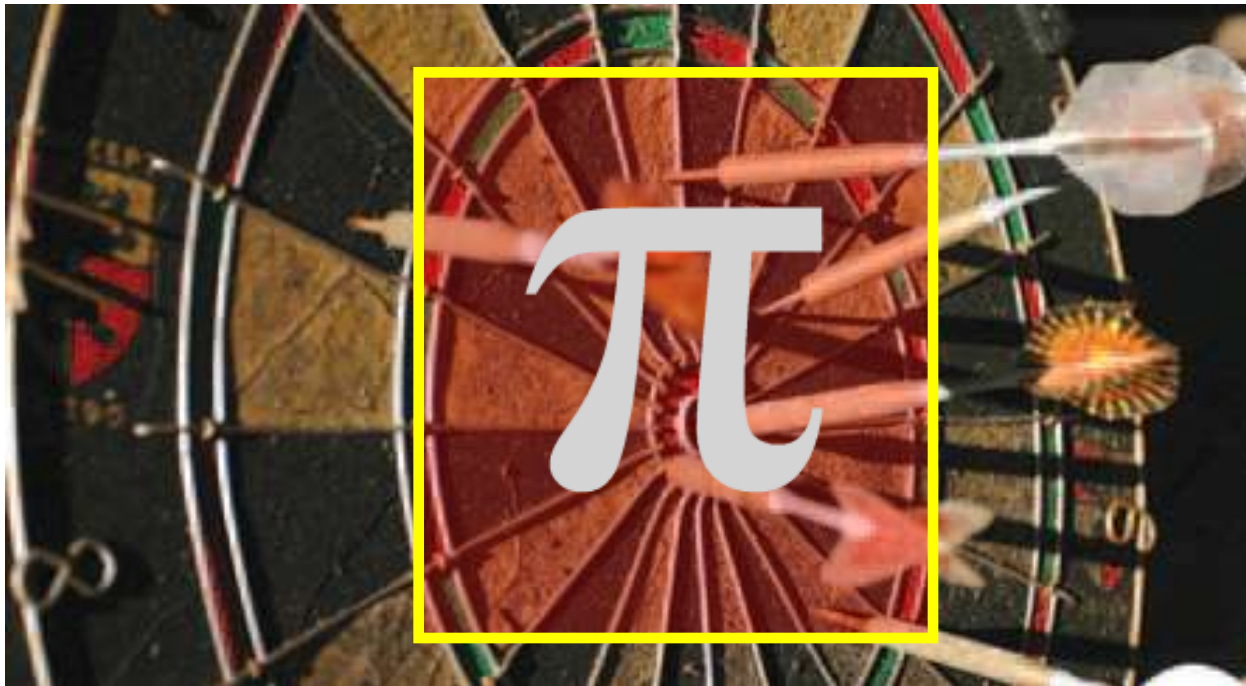
The Monte Carlo Method

While we are in this fine casino,
we should at least play one game at the “*table d’darts*”



The Monte Carlo Method

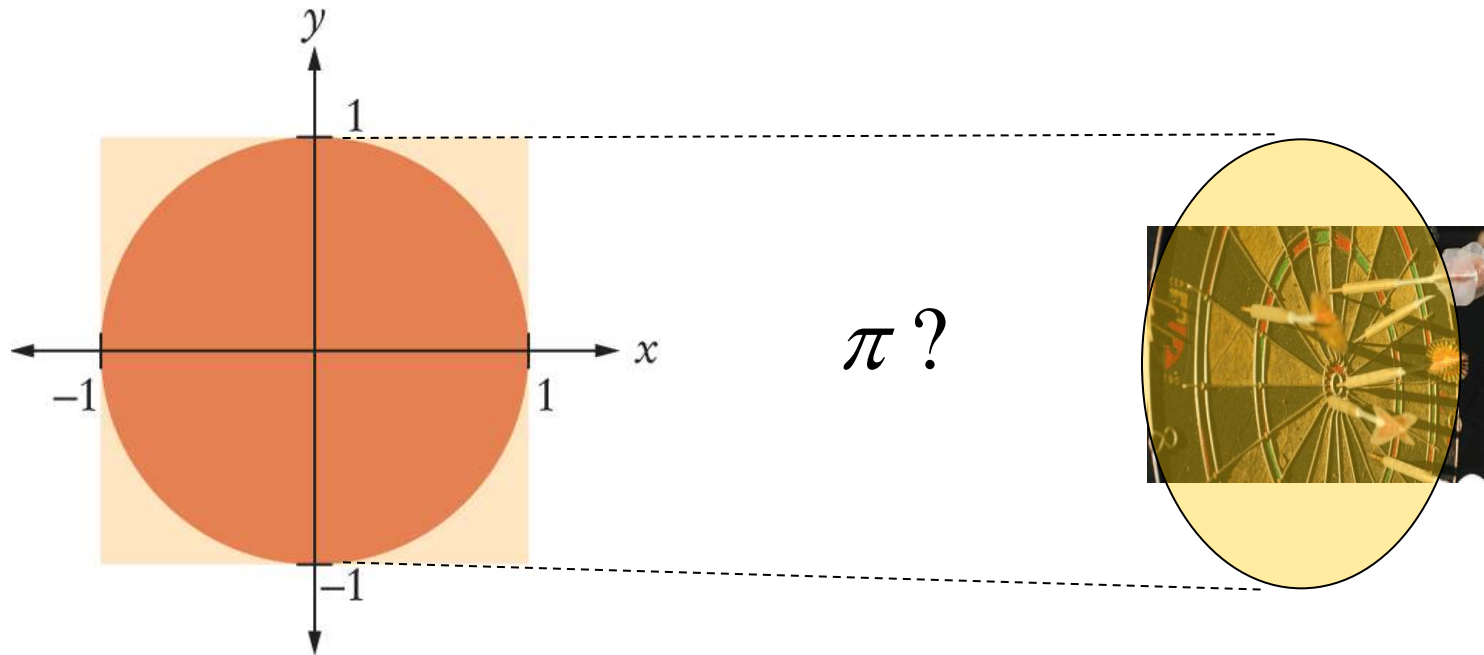
THAT'S IT!



By shooting darts (and a little math)
we can obtain an approximation for π .

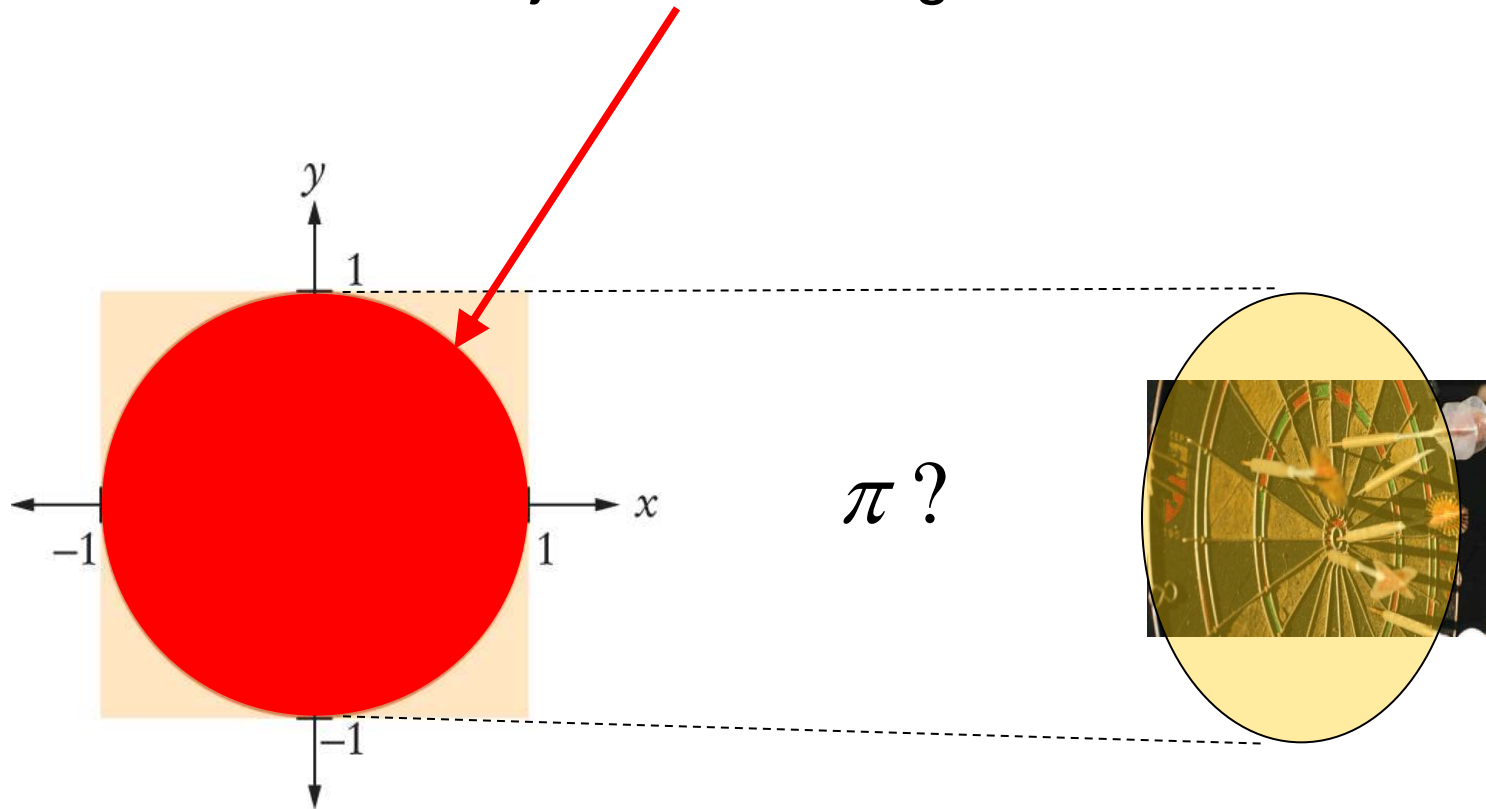
The Monte Carlo Method

Consider placing the round dartboard inside an exactly fitting square



The Monte Carlo Method

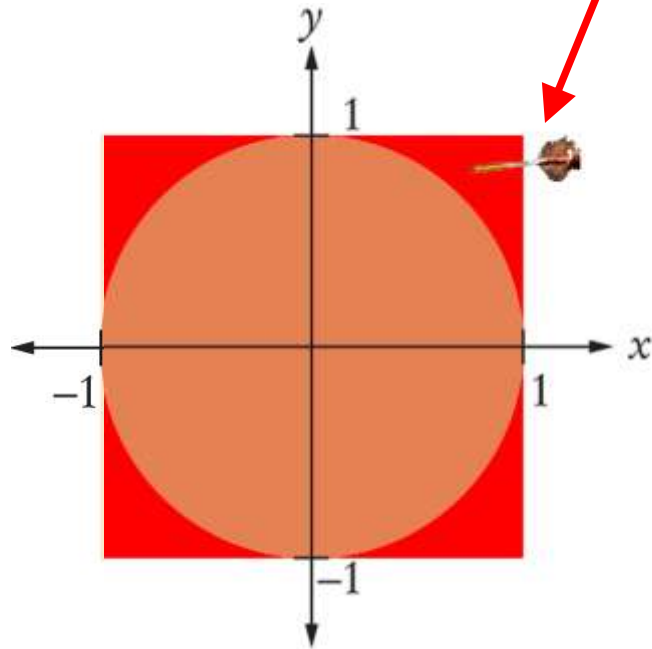
As we toss darts at the target,
if we are able to just *hit* the target – at all – it's a hit.



(no wonder this is such a pathetic casino)

The Monte Carlo Method

and a *miss* is a miss.

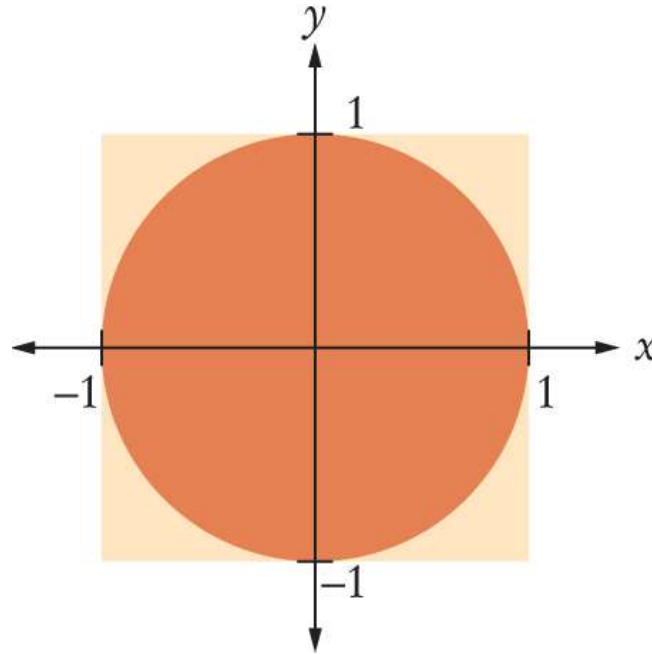


π ?

The Monte Carlo Method

The (x, y) coordinate of a *hit* is when $(x^2 + y^2) \leq 1$.
In code:

```
if (x * x + y * y <= 1) { hits++; }
```



The Monte Carlo Method

Our coded random shots will give a ratio of
hits/tries

that is approximately equal to the ratio of
the areas of the circle and the square:

$$\pi / 4$$

The Monte Carlo Method

Multiply by 4 and we have an estimate for π !

$$\pi = 4 * \text{hits/tries};$$

The longer we run our program,
the more random numbers we generate,
the better the estimate.

The Monte Carlo Method

For the x and y coordinates within the circle, we need random x and y values between -1 and 1 .

That's a range of $(-1 + 1 + 1)$ or 2 .

As before, we want to add some random portion of this range to the low endpoint, -1 .

But we will want a floating point value, not an integer.

The Monte Carlo Method

We must use `rand` with `double` values to obtain that random portion.

```
double r = rand() * 1.0 / RAND_MAX;
```

The value `r` is a random floating-point value between 0 and 1.

You can think of this as a percentage if you like.

(Use `1.0` to make the `/` operator not do integer division)

The Monte Carlo Method

The computation:

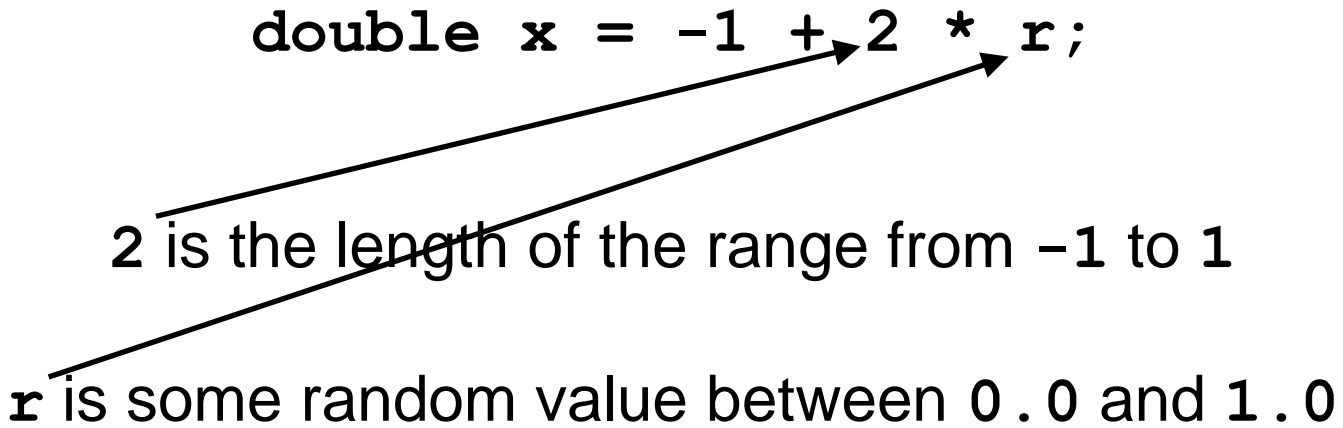
```
double x = -1 + 2 * r;
```

2 is the length of the range from **-1** to **1**

The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```




2 is the length of the range from -1 to 1

r is some random value between 0.0 and 1.0

The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```



2 is the length of the range from -1 to 1

r is some random value between 0.0 and 1.0

so $(2 * r)$ is some portion of that range



The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```

2 is the length of the range from **-1** to **1**

r is some random value between **0.0** and **1.0**

so **(2 * r)** is some portion of that range

We will add this portion to the left hand end of the range, **-1**

The Monte Carlo Method

The computation:

```
double x = -1 + 2 * r;
```

2 is the length of the range from -1 to 1

r is some random value between 0.0 and 1.0

so $(2 * r)$ is some portion of that range

Adding this portion to the left hand end of the range gives us:

x randomly within the range -1 and 1.

The Monte Carlo Method for Approximating PI

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
using namespace std;

int main()
{
    const int TRIES = 10000;
    srand(time(0));
    int hits = 0;
    for (int i = 1; i <= TRIES; i++)
    {
        double r = rand() * 1.0 / RAND_MAX; // Between 0 and 1
        double x = -1 + 2 * r; // Between -1 and 1
        r = rand() * 1.0 / RAND_MAX;
        double y = -1 + 2 * r;
        if (x * x + y * y <= 1) { hits++; }
    }
    double pi_estimate = 4.0 * hits / TRIES;
    cout << "Estimate for pi: " << pi_estimate << endl;
    return 0;
}
```

ch04/montecarlo.cpp

Chapter Summary

Explain the flow of execution in a loop.

- Loops execute a block of code repeatedly while a condition remains true.



- An off-by-one error is a common error when programming loops. Think through simple test cases to avoid this type of error.

Use the technique of hand-tracing to analyze the behavior of a program.

- Hand-tracing is a simulation of code execution in which you step through instructions and track the values of the variables.
- Hand-tracing can help you understand how an unfamiliar algorithm works.
- Hand-tracing can show errors in code or pseudocode.

Use for loops for implementing counting loops.



- The for loop is used when a value runs from a starting point to an ending point with a constant increment or decrement.

Chapter Summary

Choose between the while loop and the do loop.

- The do loop is appropriate when the loop body must be executed at least once.

Implement loops that read sequences of input data.

- A sentinel value denotes the end of a data set, but it is not part of the data.
- You can use a Boolean variable to control a loop. Set the variable to true before entering the loop, then set it to false to leave the loop.
- Use input redirection to read input from a file. Use output redirection to capture program output in a file.



Use the technique of storyboarding for planning user interactions.

- A storyboard consists of annotated sketches for each step in an action sequence.
- Developing a storyboard helps you understand the inputs and outputs that are required for a program.

Chapter Summary

Know the most common loop algorithms.

- To compute an average, keep a total and a count of all values.
- To count values that fulfill a condition, check all values and increment a counter for each match.
- If your goal is to find a match, exit the loop when the match is found.
- To find the largest value, update the largest value seen so far whenever you see a larger one.
- To compare adjacent inputs, store the preceding input in a variable.

Use nested loops to implement multiple levels of iteration.



- When the body of a loop contains another loop, the loops are nested. A typical use of nested loops is printing a table with rows and columns.

Apply loops to the implementation of simulations.

- In a simulation, you use the computer to simulate an activity. You can introduce randomness by calling the random number generator.





End Loops II